Operating System - Multi-Threading

Sangeeta Sen

Amrita Namtirtha

Department of Information Technology
National institute of Technology, Durgapur, WB

March 11, 2016

Overview

Overview

Introduction

What is Thread?

- A thread is a flow of execution through the process code, with its own program counter, system registers and stack.
- A thread is also called a light weight process.
- Threads provide a way to improve application performance through parallelism.

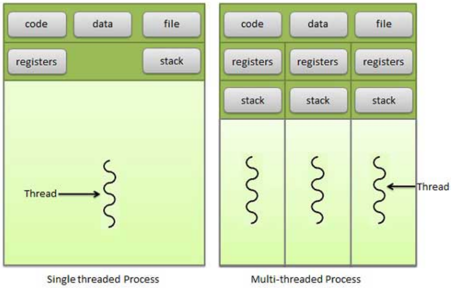

Figure: Thread processes

cont..

Advantages of Thread

- Responsiveness
- Resource sharing, hence allowing better utilization of resources.
- Economy. Creating and managing threads becomes easier.
- Scalability. One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.
- Context Switching is smooth. Context switching refers to the procedure followed by CPU to change from one task to another.

Overview

Difference between process and thread

- Process is heavy weight or resource intensive. Thread is light weight taking lesser resources than a process.
- Process switching needs interaction with operating system. Thread switching does not need to interact with operating system.
- In multiple processing environments each process executes the same code but has its own memory and file resources. All threads can share same set of open files, child processes.
- If one process is blocked then no other process can execute until the first process is unblocked. While one thread is blocked and waiting, second thread in the same task can run.
- Multiple processes without using threads use more resources. Multiple threaded processes use fewer resources.
- In multiple processes each process operates independently of the others. One thread can read, write or change another thread's data.

Overview

Start thread programming

The threading module builds on the low-level features of thread to make working with threads even easier and more pythonic. Using threads allows a program to run multiple operations concurrently in the same process space.

Thread Objects The simplest way to use a Thread is to instantiate it with a target function and call start() to let it begin working.

Simple program

```
import threading
def worker():
"""thread worker function"""
  print 'Worker'
  return
threads = []
for i in range(5):
  t = threading.Thread(target=worker)
  threads.append(t)
  t.start()
```

Cont..

Second program

```
import threading
def worker(num):
"""thread worker function"""
   print 'Worker: %s'%num
   return
threads = []
for i in range(5):
  t = threading.Thread(target=worker, args=(i,))
  threads.append(t)
  t.start()
```

Determining the Current Thread

Each Thread instance has a name with a default value that can be changed as the thread is created. Naming threads is useful in server processes with multiple service threads handling different operations.

Program

Program

```
import threading
import time
def worker():
  print threading.currentThread().getName(), 'Starting'
  time.sleep(2)
  print threading.currentThread().getName(), 'Exiting'
def my_service():
  print threading.currentThread().getName(), 'Starting'
  time.sleep(3)
  print threading.currentThread().getName(), 'Exiting'
t = threading.Thread(name='my_service', target=my_service)
w = threading.Thread(name='worker', target=worker)
w2 = threading.Thread(target=worker)
w.start()
w2.start()
t.start()
```

Overview

Another way to start thread

Program

```
import thread
import time
def print_time( threadName, delay):
  count = 0
  while count < 5:
    time.sleep(delay)
    count += 1
    print "%s : %s"% ( threadName, time.ctime(time.time()) )
try:
  thread.start_new_thread(print_time, ("Thread-1", 2, ) )
  thread.start_new_thread(print_time, ("Thread-2", 4, ) )
except:
  print "Error: unable to start thread"
while 1:
  pass
```

The Threading Module

The Threading Module

The threading module exposes all the methods of the thread module and provides some additional methods:

1. **threading.activeCount:** Returns the number of thread objects that are active.
2. **threading.currentThread:** Returns the number of thread objects in the caller's thread control.
3. **threading.enumerate:** Returns a list of all thread objects that are currently active.

Program

```
import threading
def main():
  print threading.activeCount()
  print threading.currentThread()
  print threading.enumerate()
main()
```

Overview

Life cycle of thread

```
import threading, random, time
class PrintThread( threading.Thread ):
   """Subclass of threading.Thread"""
   def __init__( self, threadName ):
     """Initialize thread, set sleep time, print data"""
     threading.Thread.__init__( self, name = threadName )
     self.sleepTime = random.randrange( 1, 6 )
     print "Name: %s;"% self.getName()
     print "sleep: %d"% self.sleepTime
   "' overridden Thread run method"'
   def run( self ):
     """Sleep for 1-5 seconds"""
     print "%sgoingtosleep"% self.getName()
     print "for %ssecond(s)"%self.sleepTime
     time.sleep( self.sleepTime )
     print self.getName(), "done sleeping"
thread1 = PrintThread( "thread1" )
thread2 = PrintThread( "thread2" )
thread3 = PrintThread( "thread3" )
print "Starting threads"
thread1.start() "'invokes run method of thread1"'
thread2.start() "' invokes run method of thread2"'
thread3.start() "' invokes run method of thread3"'
print "Threads started"
```

Overview

Synchronizing Threads

Thread Synchronization

The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads.

- A new lock is created by calling the Lock method, which returns the new lock.
- The acquire method of the new lock object is used to force threads to run synchronously.
- The release method of the new lock object is used to release the lock when it is no longer required.

Cont..

Program

```
import threading, time
class myThread (threading.Thread):
  def __init__(self, threadID, name, counter):
    threading.Thread.__init__(self)
    self.threadID = threadID
    self.name = name
    self.counter = counter
  def run(self):
    print "Starting " + self.name
    threadLock.acquire()
    print_time(self.name, self.counter, 3)
    threadLock.release()
def print_time(threadName, delay, counter):
  while counter:
    time.sleep(delay)
    print "%s : %s"%(threadName, time.ctime(time.time()))
    counter -= 1
threadLock = threading.Lock()
threads = []
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)
thread1.start()    thread2.start()
threads.append(thread1)
threads.append(thread2)
for t in threads:
  t.join()
print "Exiting Main Thread"
```

Overview

Multithreaded Priority Queue

Priority Queue

The Queue module allows you to create a new queue object that can hold a specific number of items. There are following methods to control the Queue

- get: The get removes and returns an item from the queue.
- put: The put adds item to a queue.
- qsize : The qsize returns the number of items that are currently in the queue.
- empty: The empty returns True if queue is empty; otherwise, False.
- full: the full returns True if queue is full; otherwise, False.

Program

```
import Queue,threading,time
exitFlag = 0
class myThread (threading.Thread):
   def __init__(self, threadID, name, q):
    threading.Thread.__init__(self)
    self.threadID = threadID
    self.name = name
     self.q = q
  def run(self):
    print "Starting " + self.name
     process_data(self.name, self.q)
     print "Exiting " + self.name
def process_data(threadName, q):
  while not exitFlag:
     queueLock.acquire()
     if not workQueue.empty():
        data = q.get()
       queueLock.release()
        print "%sprocessing%s"% (threadName, data)
     else:
     queueLock.release()
     time.sleep(1)
threadList = ["Thread-1", "Thread-2", "Thread-3"]
```

Program

```
nameList = ["One", "Two", "Three", "Four", "Five"]
queueLock = threading.Lock()
workQueue = Queue.Queue(10)
threads = []
threadID = 1
for tName in threadList:
    thread = myThread(threadID, tName, workQueue)
    thread.start()
    threads.append(thread)
    threadID += 1
queueLock.acquire()
for word in nameList:
    workQueue.put(word)
queueLock.release()
while not workQueue.empty():
    pass
exitFlag = 1
for t in threads:
    t.join()
print "Exiting Main Thread"
```

Thank you